Package Knapsack

Muhammad Faizan Khalid, Fahed Maqbool, Asad Enver

October 13, 2019

Contents

A Package to solve knapsack problem using 3 different approaches.

Kanpsack?

The knapsack is a discrete optimization problem where we have a sack that can filled with a limited **weight w** by adding **number of items i = 1, . . . , n,** with a **weight wi** and a **value vi** to get the maximum value. This problem is `NP-hard, meaning that it is "at least as hard as the hardest problem in NP"` ref: (via)

Three Different approaches to solve a knapsack problem

- brute force **brute force knapsack(x,w, parallel = FALSE)**
- dynamic programming **dynamic_program(x,w)**

- greedy heuristic **greedy_knapsack(x,w)**

##Parameters

- **x** is a data frame which has two columns named **w** as weight column and **v** as value of each item.
- **w** is represent a scak capcaity to the sack can be filled.
- **parallel** which is only used in brute force search alogrithm if you want to run on mulitple core than set is as **TRUE**. The default value is set as **FALSE**

##generate a data

```r
devtools::load_all()
```

```
## Loading knapsackProblems
```

```r
library("knapsackProblems")
set.seed(345)
n <- 3000
knapsack_objects <- data.frame( w=sample(1:4000, size = n, replace = TRUE),
                                v=runif(n = n, 0, 10000)
)

head(knapsack_objects)
```

```
##      w         v
## 1 1885 6822.7289
## 2 1623 8291.8846
## 3  979 7744.4478
## 4 2015 3001.8289
## 5 3989  302.1338
## 6 3506 5327.2408
```

brute_force_knapsack(x,w)

```r
brute_force_knapsack(knapsack_objects[1:8,], w = 3500)
```

```
## $value
## [1] 24289
##
## $elements
## [1] 3 7 8
```

```r
brute_force_knapsack(x = knapsack_objects[1:12,], w = 2000)
```

```
## $value
## [1] 16826
##
## $elements
## [1]  3 11
```

How long does it takes to run the algorithm for n = 16 objects?

```r
set.seed(42)
n <- 16
knapsack_objects <- data.frame( w=sample(1:4000, size = n, replace = TRUE),
                                v=runif(n = n, 0, 10000))

system.time(result1 <- brute_force_knapsack(x = knapsack_objects[1:12,], w = 2000))
```

```
##    user  system elapsed
##    0.09    0.00    0.10
```

dynamic_program(x, w)

```r
dynamic_program(knapsack_objects[1:8,], w = 3500)
```

```
## $value
## [1] 24110
##
## $elements
## [1] 7
```

```r
dynamic_program(x = knapsack_objects[1:12,], w = 2000)
```

```
## $value
## [1] 18507
##
## $elements
## [1] 7
```

How long does it takes to run the algorithm for n = 500 objects?

```r
set.seed(42)
n <- 500
knapsack_objects <- data.frame( w=sample(1:4000, size = n, replace = TRUE),
                                v=runif(n = n, 0, 10000))
system.time(result2<- dynamic_program(x = knapsack_objects[1:12,], w = 2000))
```

```
##    user  system elapsed
##     0.5     0.0     0.5
```

greedy_knapsack(x, w)

```r
greedy_knapsack(knapsack_objects[1:800,], w = 3500)
```

```
## $value
## [1] 152979
##
## $elements
## $elements[[1]]
## [1]    2.000 4136.465
##
## $elements[[2]]
## [1]   25.000 5607.079
##
## $elements[[3]]
## [1]   33.000 7148.488
##
## $elements[[4]]
## [1]   16.000 3038.528
##
## $elements[[5]]
## [1]   62.000 9628.968
##
## $elements[[6]]
## [1]   49.000 5844.997
##
## $elements[[7]]
## [1]   78.00 9145.28
##
## $elements[[8]]
## [1]   82.000 6852.282
##
## $elements[[9]]
## [1]  103.000 8336.876
##
## $elements[[10]]
## [1]   91.000 7213.336
##
## $elements[[11]]
## [1]   94.000 5764.457
##
## $elements[[12]]
## [1]  159.00 9502.03
##
## $elements[[13]]
## [1]   76.000 3410.399
##
## $elements[[14]]
## [1]  197.000 8118.797
##
## $elements[[15]]
## [1]  149.000 5860.885
##
## $elements[[16]]
## [1]  148.000 5669.993
##
## $elements[[17]]
## [1]  201.00 5532.47
##
## $elements[[18]]
## [1]  356.000 9204.891
##
## $elements[[19]]
## [1]  317.000 7963.194
##
## $elements[[20]]
## [1]   82.000 2031.335
##
## $elements[[21]]
## [1]  262.000 6340.281
##
## $elements[[22]]
## [1]  225.000 5036.876
##
## $elements[[23]]
## [1]  254.000 5050.446
##
## $elements[[24]]
## [1]  335.000 6540.679
```

```r
greedy_knapsack(x = knapsack_objects[1:12,], w = 2000)
```

```
## $value
## [1] 15117.29
##
## $elements
## $elements[[1]]
## [1]  634.000 9332.703
##
## $elements[[2]]
## [1] 1252.000 5784.584
```

How long does it takes to run the algorithm for n = 1000000 objects?

```r
set.seed(42)
n <- 1000000
knapsack_objects <- data.frame( w=sample(1:4000, size = n, replace = TRUE),
                                v=runif(n = n, 0, 10000))
system.time(result3 <- greedy_knapsack(x = knapsack_objects[1:12,], w = 2000))
```

```
##    user  system elapsed
##       0       0       0
```

Paralellism brute force search

Paralellism is implemented in Brute Force algorithm to optimize performance of program. Usage of paralellism will be achieved by setting paralell parameter to TRUE in brute fore function.

Microbenchmark package

Used for finding out the execution time of a function

```r
library(microbenchmark)

microbenchmark(
  "brute_force_knapsack"= result1, #n= 16
  "dynamic_program" = result2, #n= 500
  "greedy_knapsack" = result3 ,#n = 1000000
  times = 1,
  unit = "us"
)
```

```
## Unit: microseconds
##                  expr min  lq mean median  uq max neval
##  brute_force_knapsack 0.8 0.8  0.8    0.8 0.8 0.8     1
##       dynamic_program 2.6 2.6  2.6    2.6 2.6 2.6     1
##       greedy_knapsack 1.1 1.1  1.1    1.1 1.1 1.1     1
```